Program Promenade

連載

4平方定理

和田 英一（IIJ 技術研究所）
wada@u-tokyo.ac.jp

■Lagrangeの定理

プログラミングの楽しさをともに語ろうと始めたプロムナードだが，自分の興味本意で難かしそうな問題を取り上げがちであった．その結果，プログラムがかなり複雑になり，読む方も大変なのではないかという反省から，今回は2003年11月の会津大会の問題B, Lagrange's Four-Square Theorem (4平方定理) を話題としたい．先月に続き，素直な数学の問題である．

4平方定理とは，正の整数nは，i, j, k, lを0または正の整数として，

$$n=i^2+j^2+k^2+l^2$$

と表せるというものである．

高木貞治著す「初等整数論講義」[1]に「すべて正の整数を四つの平方数の和として表すことができる．平方数といっても0^2をも入れるのである．0を除くならば四つ以下の平方数の和というべきである」とその定理が書いてある．定理の名前はない．

同書には平方剰余を使った証明も載っているが，ここで説明するには大掛かりなのでそれは省略する．

この定理は1770年にLagrangeが証明したそうだ．コンテストの問題は定理を証明するのではなく，与えられた整数に対し，こういう表現をすべて作ってみることである．

たとえばn=25なら5^2, 4^2+3^2, $4^2+2^2+2^2+1^2$の3通りの表現がある．問題には整数n ($n<2^{15}$)が最大で255個与えられ，そのそれぞれにいく通りの表現があるか(25に対しては3)を出力する．手で計算してみると：

$1=1^2$

$2=1^2+1^2$

$3=1^2+1^2+1^2$

$4=2^2$, または$1^2+1^2+1^2+1^2$.

このように式の形で書くのは面倒だし，今回もSchemeを使うつもりなので，

```
1 (1)      2 (1 1)      3 (1 1 1)      4 (2) (1 1 1 1)
```

と降順のリスト (複数の表現があるときは，その並び) で書くことにする．この続きは次の通り．

```
5  (2 1)            12 (3 1 1 1) (2 2 2)        19 (4 1 1 1) (3 3 1)
6  (2 1 1)          13 (3 2) (2 2 2 1)          20 (4 2) (3 3 1 1)
7  (2 1 1 1)        14 (3 2 1)                  21 (4 2 1) (3 2 2 2)
8  (2 2)            15 (3 2 1 1)                22 (4 2 1 1) (3 3 2)
9  (3) (2 2 1)      16 (4) (2 2 2 2)            23 (3 3 2 1)
10 (3 1) (2 2 1 1)  17 (4 1) (3 2 2)            24 (4 2 2)
11 (3 1 1)          18 (4 1 1) (3 3) (3 2 2 1)  25 (5) (4 3) (4 2 2 1)
```

この話は実はさらに一般的で，正の整数はk角数のk個以下の和で表せるということだ．3角数とは1, 3, 6, 10, 15, …で (**図-1**) 確かに3個以下だ (上のと違い，2乗はしないで足す)．平方数は4角数のことである．

```
1  (1)          5  (3 1 1)        9  (6 3) (3 3 3) 13  (10 3) (6 6 1)
2  (1 1)        6  (6) (3 3)      10 (10) (6 3 1)  14  (10 3 1)
3  (3) (1 1 1)  7  (6 1) (3 3 1)  11 (10 1)         15  (15) (6 6 3)
4  (3 1)        8  (6 1 1)        12 (10 1 1) (6 6) (6 3 3)
```

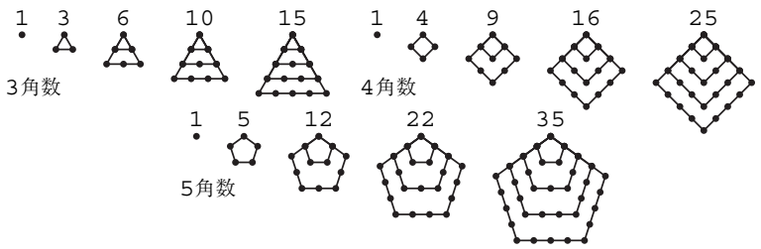

図-1

5角数なら5個以内で表せる．試みて欲しい．

■とりあえずの解き方

私のプログラミングの基本態度は，見栄えや能率化は無視し，まず答が出るところまで遮二無二書いて走らせるのである．最初はなにも見えなかった解法も，いったん動くと，不思議や見通しがよくなり，あとは見栄えと能率化に集中できる．

手による計算の実感から，n_0を表す平方数の組を計算するアルゴリズム，lag(n_0)を次のようにインフォーマルに書く(minをとるのは降順にするため)．

n_0の最大の整数平方根$\rightarrow q_0$とし
iをq_0から1まで変えながら
$n_0-i^2\rightarrow n_1$とする．n_1=0なら(i)を出力して，backup
そうでないなら
　min (n_1の最大の整数平方根，q_0)$\rightarrow q_1$とし
　jをq_1から1まで変えながら
　$n_1-j^2\rightarrow n_2$とする．n_2=0なら($i\ j$)を出力して，backup
　そうでないなら
　　min(n_2の最大の整数平方根，q_1)$\rightarrow q_2$とし
　　kをq_2から1まで変えながら
　　$n_2-k^2\rightarrow n_3$とする．n_3=0なら($i\ j\ k$)を出力して，backup
　　そうでないなら
　　　min(n_3の最大の整数平方根，q_2)$\rightarrow q_3$とし
　　　lをq_3から1まで変えながら
　　　$n_3-l^2\rightarrow n_4$とする．n_4=0なら($i\ j\ k\ l$)を出力して，backup
　　　n_4>0ならbackup

■最初のプログラム

プログラムで基本的に使う最大整数平方根(isq)と駆動ルーチン(solve)を作る．

```
(define (isq n)                                  ;⌊√n⌋
 (inexact->exact (floor (sqrt n))))              ;schemeではこう書く
(define s (open-input-file "lagrange.txt"))      ;データファイル名

(define (solve)                                  ;駆動ルーチン
 (let ((n (read s)))                             ;nを読み，lagを計算する
  (if (> n 0) (begin (newline) (display (lag n)) (solve))
              (close-input-port s))))            ;n=0ならポートを閉じる

(solve)                                          ;計算開始
```

(sqrt 10)はschemeでは3.1622776601683795となり，このfloorをとると3.となる．これを3にす

るにはさらにinexact->exactをとらなければならない．

上のスケッチに従い，整数n_0をとり，異なり表現数を返す関数を(lag n0)とする．

```
(define (lag n0)
 (let ((c 0))
  (define (disp n)                              ;出力用サブルーチン
   (set! c (+ c 1)) (display n))                ;解のリストを印字
  (let ((q0 (isq n0)))
   (do ((i q0 (- i 1))) ((< i 1))               ;iのループ
    (let* ((n1 (- n0 (* i i))) (q1 (min i (isq n1))))
     (if (= n1 0) (disp (list i))
      (do ((j q1 (- j 1))) ((< j 1))            ;jのループ
       (let* ((n2 (- n1 (* j j))) (q2 (min j (isq n2))))
        (if (= n2 0) (disp (list i j))
         (do ((k q2 (- k 1))) ((< k 1))         ;kのループ
          (let* ((n3 (- n2 (* k k))) (q3 (min k (isq n3))))
           (if (= n3 0) (disp (list i j k))
            (do ((l q3 (- l 1))) ((< l 1))  ;lのループ
             (let* ((n4 (- n3 (* l l))))
              (if (= n4 0) (disp (list i j k l))))))))))))))))c))      ;cを返す
```

これで一応満足できる答が出る．出力用のルーチンは，最終的にはcを得るだけでよいが，途中のチェックのため，リストの出力が見たいため，用意してある．完成後は(display n)をコメントアウトする．

■改良工事1

このプログラムの最初の問題点はiを正直に1まで変えていることである．n_0=16の場合の(2 2 2 2)を見れば，解の組合せのリストを降順に並べるので，iは2まで調べればよく，1は不要．変える終点はn_0の1/4の最大整数平方根までである．同様にjもn_1の1/3の最大整数平方根までよい．つまり終点r_0, r_1, r_2, r_3を計算し，doループをそこで停止するのである．

```
(let ((q0 (isq n0)) (r0 (isq (/ n0 4))))
 (do ((i q0 (- i 1))) ((< i r0))                ;iのループ，r0で止める
  (let ((n1 (- n0 (* i i))))
   (if (= n1 0) (disp (list i))
    (let ((q1 (min i (isq n1))) (r1 (isq (/ n1 3))))
     (do ((j q1 (- j 1))) ((< j r1))            ;jのループ，r1で止める
      (let ((n2 (- n1 (* j j))))
       (if (= n2 0) (disp (list i j))
        (let ((q2 (min j (isq n2))) (r2 (isq (/ n2 2))))
         (do ((k q2 (- k 1))) ((< k r2))        ;kのループ，r2で止める
          (let ((n3 (- n2 (* k k))))
           (if (= n3 0) (disp (list i j k))
            (let ((q3 (min k (isq n3))) (r3 (isq n3)))
             (do ((l q3 (- l 1))) ((< l r3));lのループ，r3で止める
              (let ((n4 (- n3 (* l l))))
               (if (= n4 0) (disp (list i j k l))))))))))))))))
```

もう一度16の例に戻れば，n_0が16なのでq_0は4，r_0は2であり，iは4から2まで調べることになる．

■改良工事2

このプログラムの始めの方をさらによく眺めてみると，iをq_0から1ずつ下げながらn_1を作り，それが0に等しいかを見ているが，等しい可能性があるのは最初の1回だけである．iが小さくなると，n_1, つまり$n_0-i\times i$は大きくなるから0から遠ざかる．それなのに毎回等価のテストをするのは馬鹿げている．

改良はiのループに入る前に$n_0=q_0\times q_0$のチェックをやることである．それが真なら(disp q0)を行い，

q_0から1を引いておく．それからr_0を計算し，iのループに入る．

一番外側のループは次のようになる．

```
(let ((q0 (isq n0)))
  (if (= n0 (* q0 q0)) (begin (disp (list q0)) (set! q0 (- q0 1))))
  (let ((r0 (isq (/ n0 4))))
    (do ((i q0 (- i 1))) ((< i r0))
      (let* ((n1 (- n0 (* i i))) (q1 (min i (isq n1))))
        ...
```

■改良工事3

iループは$i=q_0, q_0-1, q_0-2, \ldots$と変わり，$n_1$は$n_0-i^2$である．$n_1$の順次の値とその差を調べると，

$$
\begin{aligned}
&n_1=n_0-q_0^2 \\
&n_1=n_0-(q_0-1)^2=n_0-q_0^2+2q_0-1 && > 2q_0-1=2q_0-1=2i-1 \\
&n_1=n_0-(q_0-2)^2=n_0-q_0^2+4q_0-4 && > 2q_0-3=2(q_0-1)-1=2i-1 \\
&\cdots
\end{aligned}
$$

つまりn_1は初期値が$n_0-q_0^2$, 増分が$2i-1$である．このこともすべてのループのプログラムに組み込む．lのループは必要ないことも分かる．一発目でだめなら，4個以上の平方の和になってしまうからである．その改良も入れて最後のプログラムになるのだが，その前に平方根についても検討する．

■sqrtはちっとも遅くはない

このプログラムを書いていて気になったのは平方根を何回もとることである．与えられる整数が2^{15}以内なら，整数の平方根は180個程度なので，最初に1回計算し，あとは表を引くのはどうかと考えた．**図-2**は18までの整数の平方数を大きさの順に二進木に表現したものである．

81[1 18]はこの木の左端インデックスが1，要素数が18であることを示す．⌊(要素数−1)/2⌋番目が中央の位置，それに左端インデックスを足すとこの値のインデックス(つまり平方根)が得られる．

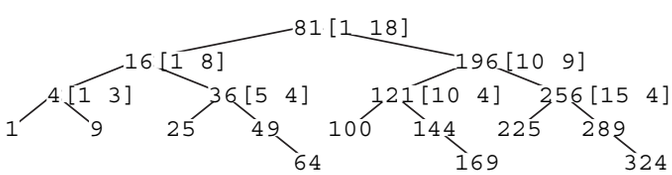

図-2

この図に対応する，実際のリストは以下のとおり．プログラムはその下に示す．

```
(81 (16 (4 (1 () ()) (9 () ()))
        (36 (25 () ()) (49 () (64 () ()))))
    (196 (121 (100 () ()) (144 () (169 () ())))
         (256 (225 () ()) (289 () (324 () ())))))
```

```
(define (isq n)
 (define (isq1 n t b l)            ;bは左端インデックス, lは要素数
  (let ((e (car t)) (ls (quotient (- l 1) 2)))    ;eは中央の値 lsは中央の位置
   (cond ((= n e) (+ ls b))        ;nとeの大小関係を調べる
         ((< n e) (if (cadr t) (isq1 n (cadr t) b ls) (- b 1)))
         ((> n e) (if (caddr t)
                  (isq1 n (caddr t) (+ b ls 1) (- l ls 1)) b)))))
 (isq1 n bt 1 18))
```

ソートされたリストからバランスした二進木構造を作るには，Sussmanらの「計算機プログラムの構造と解釈」[2]にlist->treeなる手続きがあったので，それを利用した．

残念ながらせっかくこういうデータを用意したが，実行結果は却って遅かった．この探索プログラムは解釈実行されるが，組込み関数のsqrtはコンパイルされているからであろう．

■ほぼ最終版のプログラム

以上を総合し，以下の改良プログラムが完成する．Lisp系の言語のdoループは複数の変数を回すことができるので見た目はややこしい．

```
(do ((i q0 (- i 1)) (n1 (- n0 (* q0 q0)) (+ n1 i i -1)))
  ((< i r0)) なんたら)
```

は変数iは初期値がq0, その後(- i 1)と変える；変数n1は初期値が(- n0 (* q0 q0)), その後(+ n1 i i -1)と変える；なんたらの反復は(< i r0)になったら止める；と読む．

```
(define (lag n0)
  (let ((c 0)) (define (disp n) (set! c (+ c 1)) (display n))
    (newline)
    (let ((q0 (isq n0)))
      (if (= n0 (* q0 q0)) (begin (disp (list q0)) (set! q0 (- q0 1))))
      (let ((r0 (isq (/ n0 4))))
        (do ((i q0 (- i 1)) (n1 (- n0 (* q0 q0)) (+ n1 i i -1)))
          ((< i r0))                        ;iのループ iの終端テスト
          (let ((q1 (min i (isq n1))))
            (if (= n1 (* q1 q1))            ;n1=q1*q1 のテスト
              (begin (disp (list i q1)) (set! q1 (- q1 1))))
            (let ((r1 (isq (/ n1 3))))
              (do ((j q1 (- j 1)) (n2 (- n1 (* q1 q1)) (+ n2 j j -1)))
                ((< j r1))                  ;jのループ jの終端テスト
                (let ((q2 (min j (isq n2))))
                  (if (= n2 (* q2 q2))      ;n2=q2*q2 のテスト
                    (begin (disp (list i j q2)) (set! q2 (- q2 1))))
                  (let ((r2 (isq (/ n2 2))))
                    (do ((k q2 (- k 1))     ;kのループ
                         (n3 (- n2 (* q2 q2)) (+ n3 k k -1)))
                      ((< k r2))            ;kの終端テスト
                      (let ((q3 (min k (isq n3))))
                        (if (= n3 (* q3 q3)) (disp (list i j k q3)))))))))))))) c))                  ;cを返す
```

■Cで書く

この問題はリスト構造を使わないので，簡単にC言語に書き直せる．そう思って書いたのが以下のプログラムである．変数名，関数名などは前のschemeのプログラムと同じにしてある．

```
#include<stdio.h>
#include<math.h>
#define MIN(x,y)((x)>(y)?(y):(x))

int isq(int n) {return (int)sqrt((float)n);}     /* isq関数 */

int lag(n0)
{int c=0,q0=isq(n0);
 if(n0==q0*q0){c++;q0--;}
 {int r0=isq(n0/4), n1=n0-q0*q0, i;
```

```
  for(i=q0;i>=r0;i--) {int q1=MIN(i,isq(n1));
   if(n1==q1*q1){c++;q1--;}
   {int r1=isq(n1/3), n2=n1-q1*q1, j;
    for(j=q1;j>=r1;j--) {int q2=MIN(j,isq(n2));
     if(n2==q2*q2){c++;q2--;}
     {int r2=isq(n2/2), n3=n2-q2*q2, k;
      for(k=q2;k>=r2;k--) {int q3=MIN(k,isq(n3));
       if(n3==q3*q3)c++; n3=n3+k+k-1;}} n2=n2+j+j-1;}} n1=n1+i+i-1;}}
 return(c);}

main()
{FILE *s;
 int n;
 s=fopen("lagrange.txt","r"); fscanf(s,"%d",&n);  /*nの先読み*/
 while(n>0)                          /* n>0の間 lag(n)を出力 */
 {printf(" %d ¥n",lag(n)); fscanf(s,"%d",&n);}}
```

実行してみると，インタプリタではないから，255個のデータでもあっという間に(3, 4秒で)終わり，コンテストの問題としても適切であったと思う．

■3角数と5角数の場合

座興に3角数や5角数の場合の数を計算するプログラムを書いた．スピードはどうでもよいし，ループの回数が決められないので，再帰に書いてある．

```
(define (lag n0)
 (let ((c 0))
  (define (disp n) (set! c (+ c 1)) (display (reverse n)))
  (define (pn x)                    ;x以下のk角数を返す
   (define (pn1 l)                  ;pnの下請け
    (if (< x (cadr l)) (car l) (pn1 (cdr l))))
   (pn1 '(0 1 3 6 10 15 21 28 36 45 55))) ;5角数なら数列を変える
  (define (lg n k r)                ;再帰によるループ
   (if (> k 0)
    (do ((i (pn (if r (min (car r) n) n)) (pn (- i 1)))) ((< i 1))
     (if (= i n) (disp (cons i r))
         (lg (- n i) (- k 1) (cons i r))))))
  (newline) (display n0) (lg n0 3 '())    ;5角数なら(lg n0 5 '())とする
  (display c)))
```

平方のプログラムのisqに相当するのが，内部定義してあるpnである．最後の行の呼出しのパラメタとpnの数列さえ変えれば，何角数にでも対応する．

先月号の教育用計算機センターの特集で京都大学の報告に「興味深いのは，講義においてはプログラミングの教育が相当に行われているにもかかわらず，利用者の利用傾向にはその傾向が現れていない点である」とあった．書かず嫌いなのだろうか．解くべき問題が見当たらないのだろうか．まわりに合評する仲間不在なのだろうか．理解に苦しむ．

参考文献

1) 高木貞治 : 初等整数論講義 第 2 版 , 共立出版 (1971).

2) Abelson, H., Sussman, G. J. 他 : 計算機プログラムの構造と解釈 第二版 , ピアソンエデュケーション (2000).

（平成 16 年 2 月 17 日受付）